Data Structure and Algorithm

Laboratory Activity No. 9

Queues

|  |  |
| --- | --- |
| *Submitted by:* | *Instructor:* |
| Caasi, Karl Benedict D. | Engr. Maria Rizette H. Sayo |

October 11, 2025

# Objectives

Introduction

Another fundamental data structure is the queue. It is a close “the same” of the stack, as a queue is a collection of objects that are inserted and removed according to the first-in, first-out (FIFO) principle. That is, elements can be inserted at any time, but only the element that has been in the queue the longest can be next removed.

The Queue Abstract Data Type

Formally, the queue abstract data type defines a collection that keeps objects in a sequence, where element access and deletion are restricted to the first element in the queue, and element insertion is restricted to the back of the sequence. This restriction enforces the rule that items are inserted and deleted in a queue according to the first-in, first-out (FIFO) principle. The queue abstract data type (ADT) supports the following two fundamental methods for a queue Q:

Q.enqueue(e): Add element e to the back of queue Q.

Q.dequeue( ): Remove and return the first element from queue Q;

an error occurs if the queue is empty.

The queue ADT also includes the following supporting methods (with first being analogous to the stack’s top method):

Q.first(): Return a reference to the element at the front of queue Q, without removing it; an error occurs if the queue is empty.

Q.is empty( ): Return True if queue Q does not contain any elements.

len(Q): Return the number of elements in queue Q; in Python, we implement this with the special method len .

This laboratory activity aims to implement the principles and techniques in:

* Writing Python program using Queues

Writing a Python program that will implement Queues operations

# Methods

Instruction: Type the python codes below in your Colab. Reconstruct them by implementing Queues (FIFO) algorithm. Hint: You may use Array or Linked List

# Stack implementation in python

# Creating a stack

def create\_stack():

    stack = []

    return stack

# Creating an empty stack

def is\_empty(stack):

    return len(stack) == 0

# Adding items into the stack

def push(stack, item):

    stack.append(item)

    print("Pushed Element: " + item)

# Removing an element from the stack

def pop(stack):

    if (is\_empty(stack)):

        return "The stack is empty"

    return stack.pop()

stack = create\_stack()

push(stack, str(1))

push(stack, str(2))

push(stack, str(3))

push(stack, str(4))

push(stack, str(5))

print("The elements in the stack are:"+ str(stack))

Answer the following questions:

1. What is the main difference between the stack and queue implementations in terms of element removal?
2. What would happen if we try to dequeue from an empty queue, and how is this handled in the code?
3. If we modify the enqueue operation to add elements at the beginning instead of the end, how would that change the queue behavior?
4. What are the advantages and disadvantages of implementing a queue using linked lists versus arrays?
5. In real-world applications, what are some practical use cases where queues are preferred over stacks?

# Results

Present the visualized procedures done. Also present the results with corresponding data visualizations such as graphs, charts, tables, or image . Please provide insights, commentaries, or explanations regarding the data. If an explanation requires the support of literature such as academic journals, books, magazines, reports, or web articles please cite and reference them using the IEEE format.

Please take note of the styles on the style ribbon as these would serve as the style format of this laboratory report. The body style is Times New Roman size 12, line spacing: 1.5. Body text should be in Justified alignment, while captions should be center-aligned. Images should be readable and include captions. Please refer to the sample below:

1.What is the main difference between the stack and queue implementations in terms of element removal?  
  
Answer Stack: Last In, First Out (LIFO)

2.What would happen if we try to dequeue from an empty queue, and how is this handled in the code?  
  
A black screen with text

AI-generated content may be incorrect.

3. If we modify the enqueue operation to add elements at the beginning instead of the end, how would that change the queue behavior?  
  
Answer: Add FIFO (standard queue) at the end.   
Add in the first LIFO (like a stack).

4.What are the advantages and disadvantages of implementing a queue using linked lists versus arrays?  
  
Answer: Linked list queue: more memory but greater flexibility   
Simple yet restricted capacity is what an array queue is.

5. In real-world applications, what are some practical use cases where queues are preferred over stacks?

Answer: When processing order is important, queues are utilized (first come, first served).

**CONCLUSION**I gained knowledge about Python's stacks and queues. Queues follow the FIFO rule, whereas stacks follow the LIFO rule. I gained a better understanding of data organization and error prevention by doing this exercise. It also demonstrated how these ideas relate to practical systems like memory management and task scheduling.

**REFERENCES**

Python Official Documentation: https://docs.python.org/3/tutorial/datastructures.html

W3Schools Python Data Structures: https://www.w3schools.com/python/python\_lists.asp